```
import React, { useState, useEffect, useCallback, useMemo, useRef } from "react"
import {
    addPropertyControls,
    ControlType,
    useIsOnFramerCanvas,
} from "framer"
import { motion, AnimatePresence } from "framer-motion"
import { createPortal } from "react-dom"

interface FramerResponsiveImage {
    src: string
    srcSet?: string
    alt?: string
}

interface Padding {
    top: number
    right: number
    bottom: number
    left: number
}

interface ImageWithRatio extends FramerResponsiveImage {
    aspectRatio: number
}

interface Row {
    images: ImageWithRatio[]
    height: number
}

interface JustifiedGalleryProps {
    images: FramerResponsiveImage[]
    idealRowHeight: number
    gap: number
    lastRow: "nojustify" | "justify" | "hide"
    padding: Padding | string
    style: React.CSSProperties
    enableHoverEffect: boolean
    carouselTheme: {
        backgroundColor: string
        closeButtonSize: number
        closeButtonBackground: string
```

```
        thumbnailHeight: number
        thumbnailGap: number
        thumbnailActiveScale: number
        imageBorderRadius: number
    }
}

interface CarouselProps {
    images: FramerResponsiveImage[]
    currentIndex: number
    onClose: () => void
    onIndexChange: (index: number) => void
    theme: JustifiedGalleryProps["carouselTheme"]
}

const useScrollLock = () => {
    const lockScroll = useCallback(() => {
        if (typeof window === "undefined") return
        const scrollY = window.scrollY
        document.body.style.position = "fixed"
        document.body.style.top = `-${scrollY}px`
        document.body.style.width = "100%"
        document.body.style.overflow = "hidden"
    }, [])

    const unlockScroll = useCallback(() => {
        if (typeof window === "undefined") return
        const scrollY = -parseInt(document.body.style.top || "0")
        document.body.style.position = ""
        document.body.style.top = ""
        document.body.style.width = ""
        document.body.style.overflow = ""
        window.scrollTo(0, scrollY)
    }, [])

    return { lockScroll, unlockScroll }
}

const useContainerWidth = (ref: React.RefObject<HTMLDivElement>) => {
    const [width, setWidth] = useState(0)
    useEffect(() => {
        const element = ref.current
        if (!element) return
```

```
        const observer = new ResizeObserver((entries) => {
            if (entries[0]) setWidth(entries[0].contentRect.width)
        })
        observer.observe(element)
        setWidth(element.getBoundingClientRect().width)
        return () => observer.disconnect()
    }, [ref])
    return width
}

const styles: { [key: string]: React.CSSProperties } = {
    container: {
        width: "100%",
        height: "100%",
        display: "flex",
        flexDirection: "column",
        overflow: "hidden",
    },
    canvasPlaceholder: {
        display: "flex",
        alignItems: "center",
        justifyContent: "center",
        width: "100%",
        height: "100%",
        background: "rgba(0, 153, 255, 0.1)",
        border: "1px dashed #09F",
        color: "#09F",
        fontSize: "14px",
        textAlign: "center",
        borderRadius: "8px",
    },
    row: { display: "flex", flexDirection: "row", width: "100%" },
    item: { cursor: "pointer", position: "relative", overflow: "hidden" },
    image: {
        display: "block",
        width: "100%",
        height: "100%",
        objectFit: "cover",
    },
}

const aspectRatioCache = new Map<string, number>()
```

```
export default function JustifiedGallery(props: JustifiedGalleryProps) {
    const {
        images = [],
        idealRowHeight,
        gap,
        lastRow,
        padding,
        style,
        enableHoverEffect,
        carouselTheme,
    } = props
    const isOnCanvas = useIsOnFramerCanvas()
    const containerRef = useRef<HTMLDivElement>(null)
    const containerWidth = useContainerWidth(containerRef)
    const [imageDetails, setImageDetails] = useState<ImageWithRatio[]>([])
    const [isLoading, setIsLoading] = useState(true)
    const [carouselOpen, setCarouselOpen] = useState(false)
    const [currentIndex, setCurrentIndex] = useState(0)

    useEffect(() => {
        if (!images || images.length === 0) {
            setIsLoading(false)
            setImageDetails([])
            return
        }
        setIsLoading(true)
        const loadAllImages = async () => {
            const promises = images.map(
                (img) =>
                    new Promise<ImageWithRatio>((resolve) => {
                        if (!img?.src) {
                            resolve({ ...img, aspectRatio: 1.5 })
                            return
                        }
                        if (aspectRatioCache.has(img.src)) {
                            resolve({
                                ...img,
                                aspectRatio: aspectRatioCache.get(img.src)!,
                            })
                            return
                        }
                        const imageEl = new Image()
                        imageEl.src = img.src
```

```
                imageEl.onload = () => {
                    const ratio =
                        imageEl.naturalWidth / imageEl.naturalHeight
                    aspectRatioCache.set(img.src, ratio)
                    resolve({ ...img, aspectRatio: ratio })
                }
                imageEl.onerror = () => {
                    aspectRatioCache.set(img.src, 1.5)
                    resolve({ ...img, aspectRatio: 1.5 })
                }
            })
        )
        const loadedImages = await Promise.all(promises)
        setImageDetails(loadedImages)
        setIsLoading(false)
    }
    loadAllImages()
}, [images])

const rows = useMemo<Row[]>(() => {
    if (isLoading || !containerWidth || imageDetails.length === 0) {
        return []
    }
    const calculatedRows: Row[] = []
    let currentRow: ImageWithRatio[] = []
    let currentSumOfRatios = 0
    imageDetails.forEach((img) => {
        currentRow.push(img)
        currentSumOfRatios += img.aspectRatio
        const rowWidthIfJustified =
            currentSumOfRatios * idealRowHeight +
            (currentRow.length - 1) * gap
        if (rowWidthIfJustified > containerWidth) {
            const rowHeight =
                (containerWidth - (currentRow.length - 1) * gap) /
                currentSumOfRatios
            calculatedRows.push({ images: currentRow, height: rowHeight })
            currentRow = []
            currentSumOfRatios = 0
        }
    })
    if (currentRow.length > 0) {
        if (lastRow === "hide") {
```

```
            } else if (lastRow === "justify") {
                const rowHeight =
                    (containerWidth - (currentRow.length - 1) * gap) /
                    currentSumOfRatios
                calculatedRows.push({ images: currentRow, height: rowHeight })
            } else {
                calculatedRows.push({
                    images: currentRow,
                    height: idealRowHeight,
                })
            }
        }
        return calculatedRows
    }, [containerWidth, imageDetails, idealRowHeight, gap, lastRow, isLoading])

    const containerStyle = useMemo(() => {
        let paddingValue = "0px"
        if (typeof padding === "object" && padding) {
            paddingValue = `${padding.top}px ${padding.right}px ${padding.bottom}px
${padding.left}px`
        } else if (typeof padding === "string") {
            paddingValue = padding
        }
        return { ...styles.container, ...style, padding: paddingValue }
    }, [style, padding])

    const openCarousel = (index: number) => {
        setCurrentIndex(index)
        setCarouselOpen(true)
    }
    const closeCarousel = useCallback(() => setCarouselOpen(false), [])
    const handleIndexChange = useCallback(
        (index: number) => setCurrentIndex(index),
        []
    )

    if (isOnCanvas && images.length === 0) {
        return (
            <div style={{ ...style, ...styles.canvasPlaceholder }}>
                <p>Добавьте изображения на панели свойств</p>
            </div>
        )
    }
```

```
let flatImageIndex = 0
return (
  <div ref={containerRef} style={containerStyle}>
    {isLoading && isOnCanvas && <p>Загрузка изображений...</p>}
    {rows.map((row, rowIndex) => (
      <div
        key={rowIndex}
        style={{
          ...styles.row,
          height: row.height,
          marginBottom: rowIndex === rows.length - 1 ? 0 : gap,
        }}
      >
        {row.images.map((image, imageIndex) => {
          const currentFlatIndex = flatImageIndex++
          return (
            <motion.div
              key={image.src + currentFlatIndex}
              style={{
                ...styles.item,
                width: row.height * image.aspectRatio,
                height: row.height,
                marginRight:
                  imageIndex === row.images.length - 1
                    ? 0
                    : gap,
              }}
              onClick={() => openCarousel(currentFlatIndex)}
              whileHover={
                enableHoverEffect
                  ? { scale: 1.02, zIndex: 1 }
                  : {}
              }
              transition={{
                type: "spring",
                stiffness: 400,
                damping: 20,
              }}
            >
              <img
                src={image.src}
                srcSet={image.srcSet}
```

```
                            alt={image.alt || ""}
                            style={styles.image}
                            loading="lazy"
                        />
                    </motion.div>
                )
            })}
        </div>
    ))}
    {carouselOpen && (
        <Carousel
            images={images}
            currentIndex={currentIndex}
            onClose={closeCarousel}
            onIndexChange={handleIndexChange}
            theme={carouselTheme}
        />
    )}
</div>
)
}

function Carousel({
    images,
    currentIndex,
    onClose,
    onIndexChange,
    theme,
}: CarouselProps) {
    const [hasMounted, setHasMounted] = useState(false)
    const [pageIndex, setPageIndex] = useState(currentIndex)
    const { lockScroll, unlockScroll } = useScrollLock()
    const thumbnailScrollRef = useRef<HTMLDivElement>(null)

    useEffect(() => setHasMounted(true), [])

    useEffect(() => {
        lockScroll()
        const handleKeyDown = (event: KeyboardEvent) => {
            if (event.key === "ArrowRight") {
                const nextIndex = (pageIndex + 1) % images.length
                setPageIndex(nextIndex)
                onIndexChange(nextIndex)
```

```
        } else if (event.key === "ArrowLeft") {
            const prevIndex =
                (pageIndex - 1 + images.length) % images.length
            setPageIndex(prevIndex)
            onIndexChange(prevIndex)
        } else if (event.key === "Escape") {
            onClose()
        }
    }
    window.addEventListener("keydown", handleKeyDown)
    return () => {
        unlockScroll()
        window.removeEventListener("keydown", handleKeyDown)
    }
}, [
    lockScroll,
    unlockScroll,
    pageIndex,
    images.length,
    onIndexChange,
    onClose,
])

useEffect(() => {
    setPageIndex(currentIndex)
}, [currentIndex])

useEffect(() => {
    if (thumbnailScrollRef.current) {
        const thumbnailContainer = thumbnailScrollRef.current
        const thumbnailWidth = theme.thumbnailHeight + theme.thumbnailGap
        const containerWidth = thumbnailContainer.offsetWidth
        const targetScroll =
            pageIndex * thumbnailWidth -
            containerWidth / 2 +
            thumbnailWidth / 2

        thumbnailContainer.scrollTo({
            left: Math.max(0, targetScroll),
            behavior: "smooth",
        })
    }
}, [pageIndex, theme.thumbnailHeight, theme.thumbnailGap])
```

```
const handlePageChange = useCallback(
    (index: number) => {
        setPageIndex(index)
        onIndexChange(index)
    },
    [onIndexChange]
)

const handleThumbnailClick = useCallback(
    (index: number) => {
        setPageIndex(index)
        onIndexChange(index)
    },
    [onIndexChange]
)

if (!hasMounted) return null
const portalTarget = document.body

return createPortal(
    <div
        style={{
            position: "fixed",
            top: 0,
            left: 0,
            width: "100vw",
            height: "100vh",
            background: theme.backgroundColor,
            display: "flex",
            flexDirection: "column",
            zIndex: 100000,
        }}
    >
        <div
            style={{
                position: "absolute",
                top: 0,
                left: 0,
                right: 0,
                height: "60px",
                display: "flex",
                justifyContent: "space-between",
```

```
        alignItems: "center",
        padding: "0 20px",
        zIndex: 2,
      }}
    >
      <div
        style={{
          color: "white",
          fontSize: "16px",
          fontWeight: "500",
        }}
      >
        {pageIndex + 1} из {images.length}
      </div>
      <div
        role="button"
        onClick={(e) => {
          e.stopPropagation()
          onClose()
        }}
        style={{
          width: theme.closeButtonSize,
          height: theme.closeButtonSize,
          background: theme.closeButtonBackground,
          borderRadius: "50%",
          display: "flex",
          justifyContent: "center",
          alignItems: "center",
          cursor: "pointer",
          transition: "all 0.2s ease",
        }}
        onMouseEnter={(e) => {
          e.currentTarget.style.opacity = "0.8"
        }}
        onMouseLeave={(e) => {
          e.currentTarget.style.opacity = "1"
        }}
      >
        <svg width="16" height="16" viewBox="0 0 24 24" fill="none">
          <path
            d="M6 6L18 18M6 18L18 6"
            stroke="#fff"
            strokeWidth="2"
```

```
            strokeLinecap="round"
          />
        </svg>
      </div>
    </div>

    <div
      style={{
        flex: 1,
        paddingTop: "60px",
        paddingBottom: `${theme.thumbnailHeight + 40}px`,
        position: "relative",
        overflow: "hidden",
      }}
    >
      <motion.div
        style={{
          display: "flex",
          width: `${images.length * 100}%`,
          height: "100%",
        }}
        animate={{
          x: `-${pageIndex * (100 / images.length)}%`,
        }}
        transition={{
          type: "spring",
          stiffness: 300,
          damping: 30,
        }}
        drag="x"
        dragConstraints={{
          left: -((images.length - 1) * (100 / images.length)) + "%",
          right: 0,
        }}
        onDragEnd={(event, info) => {
          const threshold = 50
          if (info.offset.x > threshold && pageIndex > 0) {
            const newIndex = pageIndex - 1
            setPageIndex(newIndex)
            onIndexChange(newIndex)
          } else if (info.offset.x < -threshold && pageIndex < images.length - 1) {
            const newIndex = pageIndex + 1
            setPageIndex(newIndex)
```

```
            onIndexChange(newIndex)
          }
        }}
      >
        {images.map((image, index) => (
          <div
            key={index}
            style={{
              width: `${100 / images.length}%`,
              height: "100%",
              display: "flex",
              justifyContent: "center",
              alignItems: "center",
              padding: "40px",
              boxSizing: "border-box",
              flexShrink: 0,
            }}
          >
            <img
              src={image.src}
              srcSet={image.srcSet}
              alt={image.alt || `Изображение ${index + 1}`}
              style={{
                maxWidth: "100%",
                maxHeight: "100%",
                width: "auto",
                height: "auto",
                objectFit: "contain",
                borderRadius: theme.imageBorderRadius,
                display: "block",
              }}
              draggable={false}
            />
          </div>
        ))}
      </motion.div>
    </div>

    <div
      style={{
        position: "absolute",
        bottom: "20px",
        left: "20px",
```

```
        right: "20px",
        height: theme.thumbnailHeight,
        display: "flex",
        justifyContent: "center",
      }}
    >
      <div
        ref={thumbnailScrollRef}
        style={{
          display: "flex",
          gap: theme.thumbnailGap,
          overflowX: "auto",
          scrollbarWidth: "none",
          // @ts-ignore
          msOverflowStyle: "none",
          WebkitScrollbar: { display: "none" },
          padding: "0 10px",
          maxWidth: "100%",
        }}
      >
        {images.map((image, index) => (
          <motion.div
            key={index}
            onClick={() => handleThumbnailClick(index)}
            style={{
              minWidth: theme.thumbnailHeight,
              height: theme.thumbnailHeight,
              borderRadius: "8px",
              overflow: "hidden",
              cursor: "pointer",
              border:
                pageIndex === index
                  ? "2px solid white"
                  : "2px solid transparent",
              boxSizing: "border-box",
            }}
            animate={{
              scale:
                pageIndex === index
                  ? theme.thumbnailActiveScale
                  : 1,
              opacity: pageIndex === index ? 1 : 0.7,
            }}
```

```
                        transition={{
                            type: "spring",
                            stiffness: 300,
                            damping: 30,
                        }}
                        whileHover={{ opacity: 1 }}
                    >
                        <img
                            src={image.src}
                            alt={image.alt || `Миниатюра ${index + 1}`}
                            style={{
                                width: "100%",
                                height: "100%",
                                objectFit: "cover",
                            }}
                            draggable={false}
                        />
                    </motion.div>
                ))}
            </div>
        </div>
    </div>,
    portalTarget
  )
}

addPropertyControls(JustifiedGallery, {
  images: {
    title: "Изображения",
    type: ControlType.Array,
    control: { type: ControlType.ResponsiveImage },
    defaultValue: [],
  },
  idealRowHeight: {
    title: "Идеальная высота ряда",
    type: ControlType.Number,
    defaultValue: 220,
    min: 50,
    max: 800,
    step: 10,
    unit: "px",
    displayStepper: true,
  },
```

```
gap: {
    title: "Отступ",
    type: ControlType.Number,
    defaultValue: 8,
    min: 0,
    max: 50,
    step: 1,
    unit: "px",
},
lastRow: {
    title: "Последний ряд",
    type: ControlType.Enum,
    options: ["nojustify", "justify", "hide"],
    optionTitles: ["По левому краю", "По ширине", "Скрыть"],
    defaultValue: "nojustify",
},
padding: {
    title: "Внутренний отступ",
    type: ControlType.Padding,
    defaultValue: "0px",
},
enableHoverEffect: {
    title: "Эффект при наведении",
    type: ControlType.Boolean,
    defaultValue: true,
    enabledTitle: "Включен",
    disabledTitle: "Выключен",
},
carouselTheme: {
    title: "Настройки карусели",
    type: ControlType.Object,
    controls: {
        backgroundColor: {
            type: ControlType.Color,
            title: "Фон",
            defaultValue: "rgba(0, 0, 0, 0.95)",
        },
        closeButtonSize: {
            type: ControlType.Number,
            title: "Размер кнопки закрытия",
            defaultValue: 40,
            min: 24,
            max: 64,
```

```
            unit: "px",
        },
        closeButtonBackground: {
            type: ControlType.Color,
            title: "Фон кнопки закрытия",
            defaultValue: "rgba(255, 255, 255, 0.2)",
        },
        thumbnailHeight: {
            type: ControlType.Number,
            title: "Высота миниатюр",
            defaultValue: 60,
            min: 40,
            max: 100,
            unit: "px",
        },
        thumbnailGap: {
            type: ControlType.Number,
            title: "Отступ между миниатюрами",
            defaultValue: 8,
            min: 4,
            max: 20,
            unit: "px",
        },
        thumbnailActiveScale: {
            type: ControlType.Number,
            title: "Масштаб активной миниатюры",
            defaultValue: 1.1,
            min: 1,
            max: 1.5,
            step: 0.1,
        },
        imageBorderRadius: {
            type: ControlType.Number,
            title: "Радиус изображения",
            defaultValue: 8,
            min: 0,
            max: 50,
            unit: "px",
        },
    },
  },
})
```